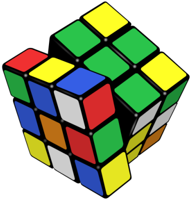
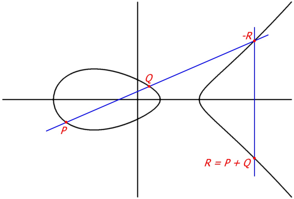
From Rubik's to cryptography

A tour of computational challenges in the field

Christophe Petit





Mary Stuart, Queen of Scots

- Born on Dec 8th, 1542
- Queen of Scots on Dec 14th

Mary Stuart, Queen of Scots

- Born on Dec 8th, 1542
- Queen of Scots on Dec 14th
- 1558 : marries François II of France, who dies in 1560
- 1565 : marries Lord Darnley, who is murdered in 1567

Mary Stuart, Queen of Scots

- Born on Dec 8th, 1542
- Queen of Scots on Dec 14th
- 1558 : marries François II of France, who dies in 1560
- 1565 : marries Lord Darnley, who is murdered in 1567
- 1567 : marries James Hepburn
- 1567 : forced to abdicate, she **flies to England**

The Babington Plot

- Mary is **made captive** by her cousin **Queen Elisabeth**
- Contacted by Babington to **conspire** against Queen Elisabeth
- They **encipher** their correspondence to keep it secret

The Babington Plot

- Mary is **made captive** by her cousin **Queen Elisabeth**
- Contacted by Babington to **conspire** against Queen Elisabeth
- They **encipher** their correspondence to keep it secret
- Conspiracy suspected but Queen Elisabeth needs proofs

A good cipher or the Death

- Principal secretary **Walsingham**, also chief of intelligence services, puts **Thomas Phelippes** on duty to **break Mary's code**

A good cipher or the Death

- Principal secretary **Walsingham**, also chief of intelligence services, puts **Thomas Phelippes** on duty to **break Mary's code**
- Mary's life now relies on the strength of her cipher. . .

Outline

Elliptic curve cryptography

Hash functions and the Rubik's cube

Side-channel attacks

Outline

Elliptic curve cryptography

Hash functions and the Rubik's cube

Side-channel attacks

Cryptography

- *Cryptos* = secret, hidden ; *graphein* = writing
- Securing communication in the presence of *adversaries*
  - Confidentiality
  - Data integrity
  - Authentication
  - Non-repudiation

Cryptography

- *Cryptos* = secret, hidden ; *graphein* = writing
- Securing communication in the presence of *adversaries*
  - Confidentiality
  - Data integrity
  - Authentication
  - Non-repudiation
- Building blocks : encryption, MACs, signature, ...

Cryptography

- *Cryptos* = secret, hidden ; *graphein* = writing
- Securing communication in the presence of *adversaries*
  - Confidentiality
  - Data integrity
  - Authentication
  - Non-repudiation
- Building blocks : encryption, MACs, signature, ...
- Many applications today : ATM cards, computer passwords, electronic commerce, electronic voting,...

Cryptography Wall of Fame

- Julius Caesar
- Abu al-Kindi
- Blaise de Vigenère
- Charles Babagge
- Auguste Kerckhoffs (ULG !)
- Claude Shannon
- Alan Turing
- Whitfield Diffie and Martin Hellman
- Ronald Rivest, Adi Shamir and Leonard Adleman
- Neal Koblitz and Victor Miller
- ...

How to "prove" security ?

- In cryptography, proofs are never absolute
- Typical theorem :

  If **Computational problem A** *is hard,*
  then **Attack B** *against* **protocol C** *is hard as well*

How to “prove” security ?

- In cryptography, proofs are never absolute
- Typical theorem :

  If **Computational problem A** *is hard,*
  then **Attack B** *against* **protocol C** *is hard as well*

- Pro : can focus on studying Problem A

How to “prove” security ?

- In cryptography, proofs are never absolute
- Typical theorem :

  If **Computational problem A** *is hard,*
  then **Attack B** *against* **protocol C** *is hard as well*

- Pro : can focus on studying Problem A
- Contra : only considers Attack B
- Contra : only meaningful if Problem A is hard

How to “prove” security ?

- In cryptography, proofs are never absolute
- Typical theorem :

  If **Computational problem A** *is hard,*
  then **Attack B** *against* **protocol C** *is hard as well*

- Pro : can focus on studying Problem A
- Contra : only considers Attack B
- Contra : only meaningful if Problem A is hard
- Good news : some computational problems seem hard

Popular cryptographic “hard problems”

- **Integer factorization** (IFP)
  Given $n = pq$ where p and q are two large primes, find p and q

Popular cryptographic "hard problems"

- **Integer factorization** (IFP)
  Given $n = pq$ where p and q are two large primes, find p and q
- **Discrete logarithm** (DLP)
  Given a large prime p, given $g, h < p$, find k such that $h = g^k \bmod p$

Popular cryptographic “hard problems”

- **Integer factorization** (IFP)
  Given $n = pq$ where p and q are two large primes,
  find p and q
- **Discrete logarithm** (DLP)
  Given a large prime p, given $g, h < p$,
  find k such that $h = g^k \bmod p$
- **Elliptic curve discrete logarithm** (ECDLP)
  Similar as DLP but multiplicative group of a finite field replaced by group of points of an elliptic curve (see below)

Additional cryptographic assumptions

- **AES** is a "good pseudorandom permutation"

Additional cryptographic assumptions

- **AES** is a "good pseudorandom permutation"
- **SHA-2** is a "good hash function" (see below)

Additional cryptographic assumptions

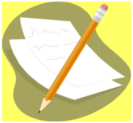
- **AES** is a “good pseudorandom permutation”
- **SHA-2** is a “good hash function” (see below)
- Lattice problems, coding theory problems, solving polynomial systems of equations

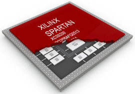
Additional cryptographic assumptions

- **AES** is a "good pseudorandom permutation"
- **SHA-2** is a "good hash function" (see below)
- Lattice problems, coding theory problems, solving polynomial systems of equations
- Many variants of previous problems
- ...

Strength of the assumptions

- Some are stronger than others
- Depends on the **size** of parameters
- Evaluated based on
  - Best algorithms
  - Computing power
  - Fame of the problem
- See `www.keylength.com`





Protocol example : Diffie-Hellman key exchange

- Mary and Babington exchange messages that can be seen by Elisabeth. They want to **share a secret key** K_{MB}

Protocol example : Diffie-Hellman key exchange

- Mary and Babington exchange messages that can be seen by Elisabeth. They want to **share a secret key** K_{MB}
  - They agree on a prime number p and on $g < p$

Protocol example : Diffie-Hellman key exchange

- Mary and Babington exchange messages that can be seen by Elisabeth. They want to **share a secret key** K_{MB}
  - They agree on a prime number p and on $g < p$
  - Mary sends $h_m := g^m \bmod p$ for random m
  - Babington sends $h_b := g^b \bmod p$ for random b

Protocol example : Diffie-Hellman key exchange

- Mary and Babington exchange messages that can be seen by Elisabeth. They want to **share a secret key** K_{MB}
  - They agree on a prime number p and on $g < p$
  - Mary sends $h_m := g^m \bmod p$ for random m
  - Babington sends $h_b := g^b \bmod p$ for random b
  - Mary computes $K_{MB} := h_b^m \bmod p$
  - Babington computes $K_{BM} := h_m^b \bmod p$

Protocol example : Diffie-Hellman key exchange

- Mary and Babington exchange messages that can be seen by Elisabeth. They want to **share a secret key** K_{MB}
  - They agree on a prime number p and on $g < p$
  - Mary sends $h_m := g^m \bmod p$ for random m
  - Babington sends $h_b := g^b \bmod p$ for random b
  - Mary computes $K_{MB} := h_b^m \bmod p$
  - Babington computes $K_{BM} := h_m^b \bmod p$
- We have $K_{MB} = g^{bm} \bmod p = g^{mb} \bmod p = K_{BM}$

Protocol example : Diffie-Hellman key exchange

- Mary and Babington exchange messages that can be seen by Elisabeth. They want to **share a secret key** K_{MB}
  - They agree on a prime number p and on $g < p$
  - Mary sends $h_m := g^m \bmod p$ for random m
  - Babington sends $h_b := g^b \bmod p$ for random b
  - Mary computes $K_{MB} := h_b^m \bmod p$
  - Babington computes $K_{BM} := h_m^b \bmod p$
- We have $K_{MB} = g^{bm} \bmod p = g^{mb} \bmod p = K_{BM}$
- Recovering m from $g^m \bmod p$ (or b from $g^b \bmod p$) is the **discrete logarithm problem**

Symmetric key vs. Public key

- In **symmetric key** cryptography,
  single secret key shared between sender and receiver
- In **public key** cryptography, one key is public, but only one person knows corresponding secret key
  - Everybody can encrypt, only one can decrypt
  - Only one can sign, everybody can check the signature

Symmetric key vs. Public key

- In **symmetric key** cryptography,
  single secret key shared between sender and receiver
- In **public key** cryptography, one key is public, but only one person knows corresponding secret key
  - Everybody can encrypt, only one can decrypt
  - Only one can sign, everybody can check the signature
- **Key management** harder for symmetric keys
- Symmetric key algorithms often more **efficient**
- Public key algorithms rely on "simpler and nicer" **complexity assumptions**

General advices to Mary Stuart

- Don't build your own algorithm !
  - AES, SHA-2, RSA, (EC)DSA well-studied

General advices to Mary Stuart

- Don't build your own algorithm !
  - AES, SHA-2, RSA, (EC)DSA well-studied
- Combine the power of symmetric and public key crypto
  - Key management easier with public key
  - Secret key algorithms more efficient
  - Use long term public keys to derive session secret keys

General advices to Mary Stuart

- Don't build your own algorithm !
  - AES, SHA-2, RSA, (EC)DSA well-studied
- Combine the power of symmetric and public key crypto
  - Key management easier with public key
  - Secret key algorithms more efficient
  - Use long term public keys to derive session secret keys
- Beware of authentication issues !
  - Textbook Diffie-Hellman can be broken with a simple man-in-the-middle attack
  - Use certificates to authenticate public keys

Elliptic curve cryptograpy

- Diffie-Hellman (and many other protocols) first described for the group $\mathbb{F}_p^*$

Elliptic curve cryptograpy

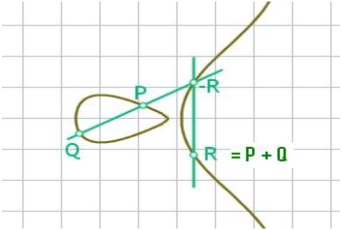
- Diffie-Hellman (and many other protocols) first described for the group $\mathbb{F}_p^*$
- 1985 : Koblitz and Miller independently proposed to use the **group of points of an elliptic curve** instead

Elliptic curves

- Points (x, y) satisfying an equation $y^2 = x^3 + Ax + B$
  Can be defined over any field K

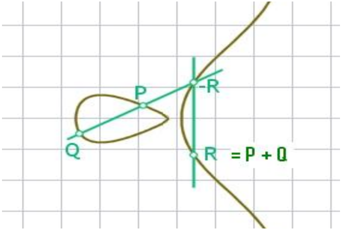
Elliptic curves

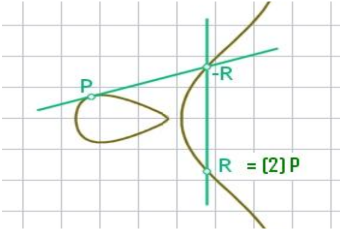
- Points (x, y) satisfying an equation $y^2 = x^3 + Ax + B$
  Can be defined over any field K
- Form an **Abelian group**



Elliptic curves

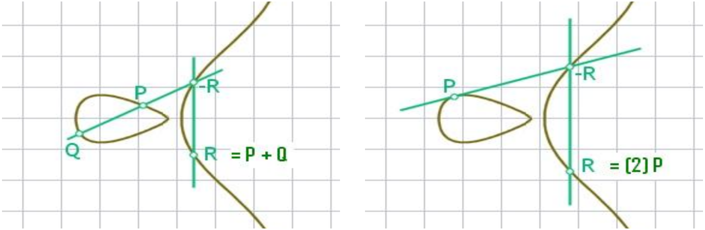
- Points (x, y) satisfying an equation $y^2 = x^3 + Ax + B$
  Can be defined over any field K
- Form an **Abelian group**





Elliptic curves

- Points (x, y) satisfying an equation $y^2 = x^3 + Ax + B$
  Can be defined over any field K
- Form an **Abelian group**



- **Elliptic curve discrete logarithm problem**
  Given P and $Q = [k]P$, find k

ECDLP vs DLP or Factoring

- ECDLP **much harder** than DLP or factoring
  - We have much better algorithms for DLP and factoring than for ECDLP
  - 1300-bit RSA or DL $\approx$ 160-bit ECDLP

ECDLP vs DLP or Factoring

- ECDLP **much harder** than DLP or factoring
  - We have much better algorithms for DLP and factoring than for ECDLP
  - 1300-bit RSA or DL $\approx$ 160-bit ECDLP
- Group addition is now rather efficient

ECDLP vs DLP or Factoring

- ECDLP **much harder** than DLP or factoring
  - We have much better algorithms for DLP and factoring than for ECDLP
  - 1300-bit RSA or DL $\approx$ 160-bit ECDLP
- Group addition is now rather efficient
- Elliptic curves offer additional features

ECDLP vs DLP or Factoring

- ECDLP **much harder** than DLP or factoring
  - We have much better algorithms for DLP and factoring than for ECDLP
  - 1300-bit RSA or DL $\approx$ 160-bit ECDLP
- Group addition is now rather efficient
- Elliptic curves offer additional features
- 2000 : 15 curves recommended by NIST in FIPS 186-2

ECDLP vs DLP or Factoring

- ECDLP **much harder** than DLP or factoring
  - We have much better algorithms for DLP and factoring than for ECDLP
  - 1300-bit RSA or DL $\approx$ 160-bit ECDLP
- Group addition is now rather efficient
- Elliptic curves offer additional features
- 2000 : 15 curves recommended by NIST in FIPS 186-2
- 2009 : NSA advocates use of ECC

A theoretical breakthrough

- 30-year old **subexponential algorithms** for DLP and factoring, now $\approx \exp\left(c(\log|G|)^{1/3}(\log\log|G|)^{2/3}\right)$

A theoretical breakthrough

- 30-year old **subexponential algorithms** for DLP and factoring, now $\approx \exp\left(c(\log|G|)^{1/3}(\log\log|G|)^{2/3}\right)$
- Except for special curves, ECDLP remained **exponential** Best attacks were generic attacks in $\approx \exp\left((\log|G|)/2\right)$

A theoretical breakthrough

- 30-year old **subexponential algorithms** for DLP and factoring, now $\approx \exp\left(c(\log|G|)^{1/3}(\log\log|G|)^{2/3}\right)$
- Except for special curves, ECDLP remained **exponential** Best attacks were generic attacks in $\approx \exp\left((\log|G|)/2\right)$
- 2012 : **binary** curves ECDLP **subexponential** [FPPR12,PQ12]
  - Complexity $\approx \exp\left(c'(\log|G|)^{2/3}(\log|G|)\right)$

A theoretical breakthrough

- 30-year old **subexponential algorithms** for DLP and factoring, now $\approx \exp\left(c(\log|G|)^{1/3}(\log\log|G|)^{2/3}\right)$
- Except for special curves, ECDLP remained **exponential** Best attacks were generic attacks in $\approx \exp\left((\log|G|)/2\right)$
- 2012 : **binary** curves ECDLP **subexponential** [FPPR12,PQ12]
  - Complexity $\approx \exp\left(c'(\log|G|)^{2/3}(\log|G|)\right)$
  - 10/15 NIST curves are binary curves

A theoretical breakthrough

- 30-year old **subexponential algorithms** for DLP and factoring, now $\approx \exp\left(c(\log|G|)^{1/3}(\log\log|G|)^{2/3}\right)$
- Except for special curves, ECDLP remained **exponential** Best attacks were generic attacks in $\approx \exp\left((\log|G|)/2\right)$
- 2012 : **binary** curves ECDLP **subexponential** [FPPR12,PQ12]
  - Complexity $\approx \exp\left(c'(\log|G|)^{2/3}(\log|G|)\right)$
  - 10/15 NIST curves are binary curves
  - Beats Pollard rho for "large" parameters

A theoretical breakthrough

- 30-year old **subexponential algorithms** for DLP and factoring, now $\approx \exp\left(c(\log|G|)^{1/3}(\log\log|G|)^{2/3}\right)$
- Except for special curves, ECDLP remained **exponential** Best attacks were generic attacks in $\approx \exp\left((\log|G|)/2\right)$
- 2012 : **binary** curves ECDLP **subexponential** [FPPR12,PQ12]
  - Complexity $\approx \exp\left(c'(\log|G|)^{2/3}(\log|G|)\right)$
  - 10/15 NIST curves are binary curves
  - Beats Pollard rho for "large" parameters
  - Index calculus algorithm

A generic DLP algorithm : Pollard's rho

- Let P and $Q = [k]P$ in a group G. We want to find k

A generic DLP algorithm : Pollard's rho

- Let P and $Q = [k]P$ in a group G. We want to find k
- Define a "pseudorandom" function f such that $f(R)$ is either $[2]R$, $(R+S)$ or $(R+T)$

A generic DLP algorithm : Pollard's rho

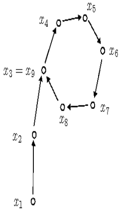
- Let P and $Q = [k]P$ in a group G. We want to find k
- Define a "pseudorandom" function f such that $f(R)$ is either $[2]R$, $(R + S)$ or $(R + T)$
- Start from $P_0 := O$ and iterate f

A generic DLP algorithm : Pollard's rho

- Let P and $Q = [k]P$ in a group G. We want to find k
- Define a "pseudorandom" function f such that $f(R)$ is either $[2]R$, $(R+S)$ or $(R+T)$
- Start from $P_0 := O$ and iterate f
- Store $P_i = [a_i]P + [b_i]Q$

A generic DLP algorithm : Pollard's rho

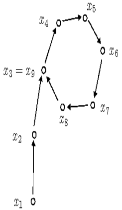
- Let P and $Q = [k]P$ in a group G. We want to find k
- Define a "pseudorandom" function f such that $f(R)$ is either $[2]R$, $(R+S)$ or $(R+T)$
- Start from $P_0 := O$ and iterate f
- Store $P_i = [a_i]P + [b_i]Q$
- When a **collision** $P_m = P_n$ is found, Deduce $Q = \left[\frac{a_m - a_n}{b_m - b_n}\right] P$



A generic DLP algorithm : Pollard's rho

- Let P and $Q = [k]P$ in a group G. We want to find k

- Define a "pseudorandom" function f such that $f(R)$ is either $[2]R$, $(R+S)$ or $(R+T)$
- Start from $P_0 := O$ and iterate f
- Store $P_i = [a_i]P + [b_i]Q$
- When a **collision** $P_m = P_n$ is found,
  Deduce $Q = \left[\frac{a_m - a_n}{b_m - b_n}\right] P$



- Time complexity $\approx |G|^{1/2} \Rightarrow$ today we need $|G| > 2^{160}$

Index calculus

- General method to solve discrete logarithm problems

Index calculus

- General method to solve discrete logarithm problems
  1. Define a **factor basis** $\mathcal{F} \subset G$

Index calculus

- General method to solve discrete logarithm problems
  1. Define a **factor basis** $\mathcal{F} \subset G$
  2. **Relation search** : find about $|\mathcal{F}|$ **relations**

$$a_i P + b_i Q = \sum_{P_j \in \mathcal{F}} e_{ij} P_j$$

Index calculus

- General method to solve discrete logarithm problems
  1. Define a **factor basis** $\mathcal{F} \subset G$
  2. **Relation search** : find about $|\mathcal{F}|$ **relations**

$$a_i P + b_i Q = \sum_{P_j \in \mathcal{F}} e_{ij} P_j$$

  3. Do **linear algebra** modulo $|G|$ on the relations to get

$$aP + bQ = 0$$

Index calculus

- General method to solve discrete logarithm problems
  1. Define a **factor basis** $\mathcal{F} \subset G$
  2. **Relation search** : find about $|\mathcal{F}|$ **relations**

  $$a_i P + b_i Q = \sum_{P_j \in \mathcal{F}} e_{ij} P_j$$

  3. Do **linear algebra** modulo $|G|$ on the relations to get

  $$aP + bQ = 0$$

- Success depends on the group

Index calculus

- General method to solve discrete logarithm problems
  1. Define a **factor basis** $\mathcal{F} \subset G$
  2. **Relation search** : find about $|\mathcal{F}|$ **relations**
  $$a_i P + b_i Q = \sum_{P_j \in \mathcal{F}} e_{ij} P_j$$
  3. Do **linear algebra** modulo $|G|$ on the relations to get
  $$aP + bQ = 0$$
- Success depends on the group
- Can be adapted for factoring

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$

- **Relation search**
  - Choose random $a, b \in \{1, \ldots, p-1\}$
  - Compute $r := g^a h^b \bmod p$

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$

- **Relation search**
  - Choose random $a, b \in \{1, \ldots, p-1\}$
  - Compute $r := g^a h^b \bmod p$
  - If all factors of r are $\leq B$, store a relation $[a]g + [b]h = \sum_{p_i \in \mathcal{F}_B} [e_i] p_i$

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$

- **Relation search**
  - Choose random $a, b \in \{1, \ldots, p-1\}$
  - Compute $r := g^a h^b \bmod p$
  - If all factors of r are $\leq B$, store a relation $[a]g + [b]h = \sum_{p_i \in \mathcal{F}_B} [e_i] p_i$
- **Linear algebra** modulo $p - 1$ on the relations

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$

- **Relation search**
  - Choose random $a, b \in \{1, \ldots, p-1\}$
  - Compute $r := g^a h^b \bmod p$
  - If all factors of r are $\leq B$, store a relation $[a]g + [b]h = \sum_{p_i \in \mathcal{F}_B} [e_i] p_i$
- **Linear algebra** modulo $p-1$ on the relations
- For $B \approx \exp((\log p)^{1/2})$, **subexponential complexity**

Index calculus in practice

- Relation search is distributed
  - Can use FPGAs, graphic cards, playstations, cloud computing...
  - RSA-768 factorization : 2000 computer cores years

Index calculus in practice

- Relation search is distributed
  - Can use FPGAs, graphic cards, playstations, cloud computing. . .
  - RSA-768 factorization : 2000 computer cores years
- Linear algebra is not trivial
  - Memory may be larger constraint than time
  - Preprocessing, block algorithms, sparse algorithms,. . .
  - RSA-768 factorization : 252.735.215 square matrix with 14.7 non-zero entries per row on average

Index calculus in practice

- Relation search is distributed
  - Can use FPGAs, graphic cards, playstations, cloud computing. . .
  - RSA-768 factorization : 2000 computer cores years
- Linear algebra is not trivial
  - Memory may be larger constraint than time
  - Preprocessing, block algorithms, sparse algorithms,. . .
  - RSA-768 factorization : 252.735.215 square matrix with 14.7 non-zero entries per row on average
- Main costs include power and building costs. . .

My personal advice to Mary Stuart

- Elliptic curves are smaller, faster, cuter

My personal advice to Mary Stuart

- Elliptic curves are smaller, faster, cuter
- **BUT** there is a new attack on binary curves
- Practical impact still unclear
  - Could remain theoretical
  - Improvements might break current parameters
  - Could be extended to prime field elliptic curves

My personal advice to Mary Stuart

- Elliptic curves are smaller, faster, cuter
- **BUT** there is a new attack on binary curves
- Practical impact still unclear
  - Could remain theoretical
  - Improvements might break current parameters
  - Could be extended to prime field elliptic curves
- Avoid binary curves for at least five years
- Beware that algorithm improvements are more likely to come for ECDLP than DLP or factoring

Outline

Elliptic curve cryptography

Hash functions and the Rubik's cube

Side-channel attacks

Cryptographic hash functions

- "Compressing" functions

$$H : \{0,1\}^* \to \{0,1\}^n$$

Cryptographic hash functions

- "Compressing" functions

$$H : \{0,1\}^* \to \{0,1\}^n$$

- Main security properties
  - **Collision resistance :**
    hard to find m, m' such that $H(m) = H(m')$

Cryptographic hash functions

- "Compressing" functions

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

- Main security properties
  - **Collision resistance :**
    hard to find m, m' such that $H(m) = H(m')$
  - **Preimage resistance :**
    given h, hard to find m such that $H(m) = h$

Cryptographic hash functions

- "Compressing" functions

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

- Main security properties
  - **Collision resistance :**
    hard to find m, m' such that $H(m) = H(m')$
  - **Preimage resistance :**
    given h, hard to find m such that $H(m) = h$
  - **Second preimage resistance :**
    given m, hard to find m' such that $H(m') = h$

Cryptographic hash functions

- "Compressing" functions

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

- Main security properties
  - **Collision resistance :**
    hard to find m, m' such that $H(m) = H(m')$
  - **Preimage resistance :**
    given h, hard to find m such that $H(m) = h$
  - **Second preimage resistance :**
    given m, hard to find m' such that $H(m') = h$
- Often used as "pseudo-random functions"

Applications

- Message authentication codes
- Digital signatures
- Password storage
- Pseudorandom number generation
- Entropy extraction
- Key derivation techniques
- ...
- ...

Popular hash algorithms

- MD5, SHA-1, RIPEMD-128, GOST, SHA-2, SHA-3

Popular hash algorithms

- MD5, SHA-1, RIPEMD-128, GOST, SHA-2, SHA-3
- MD5 is dead !
  - 1996 : first weaknesses, shift to SHA-1 recommended
  - 2004 : first actual collisions
  - 2005 : Nostradamus attack
  - 2008 : fake root CA certificates
  - 2012 : still widely used

Popular hash algorithms

- MD5, SHA-1, RIPEMD-128, GOST, SHA-2, SHA-3
- MD5 is dead !
  - 1996 : first weaknesses, shift to SHA-1 recommended
  - 2004 : first actual collisions
  - 2005 : Nostradamus attack
  - 2008 : fake root CA certificates
  - 2012 : still widely used
- 2005 : Security of SHA-1 questionned

Popular hash algorithms

- MD5, SHA-1, RIPEMD-128, GOST, SHA-2, SHA-3
- MD5 is dead !
  - 1996 : first weaknesses, shift to SHA-1 recommended
  - 2004 : first actual collisions
  - 2005 : Nostradamus attack
  - 2008 : fake root CA certificates
  - 2012 : still widely used
- 2005 : Security of SHA-1 questionned
- 2012 : SHA-3 selected after public competition

Popular hash algorithms

- MD5, SHA-1, RIPEMD-128, GOST, SHA-2, SHA-3
- MD5 is dead !
  - 1996 : first weaknesses, shift to SHA-1 recommended
  - 2004 : first actual collisions
  - 2005 : Nostradamus attack
  - 2008 : fake root CA certificates
  - 2012 : still widely used
- 2005 : Security of SHA-1 questionned
- 2012 : SHA-3 selected after public competition
- All of them have “block cipher-like strucure”

Hash functions from Cayley graphs

- **Goal** : relate main security properties of a hash function to "simple" hard problems from **group/graph theory**

Hash functions from Cayley graphs

- **Goal** : relate main security properties of a hash function to "simple" hard problems from **group/graph theory**

- **Parameters** G a group, and $S = \{s_0, ..., s_{k-1}\} \subset G$
- Write $m = m_1 m_2 ... m_N$ with $m_i \in \{0, ..., k-1\}$
  Define
  $$H(m) := s_{m_1} s_{m_2} \ldots s_{m_N}$$

Hash functions from Cayley graphs

- **Goal** : relate main security properties of a hash function to "simple" hard problems from **group/graph theory**

- **Parameters** G a group, and $S = \{s_0, ..., s_{k-1}\} \subset G$
- Write $m = m_1 m_2 ... m_N$ with $m_i \in \{0, ..., k-1\}$
  Define
  $$H(m) := s_{m_1} s_{m_2} \ldots s_{m_N}$$

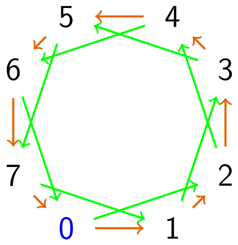
- Efficiency can be good, depending on G and S
- Parallelism : $H(m||m') = H(m)H(m')$

Cayley graph perspective

- Hash computation $\sim$ *walk* in the *Cayley graph*

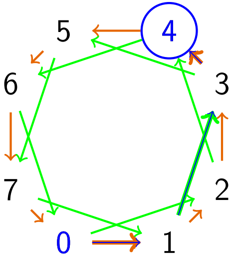
Cayley graph perspective

- Hash computation $\sim$ *walk* in the *Cayley graph*
- Example : $G = (\mathbb{Z}/8\mathbb{Z}, +)$, $S = \{1, 2\}$



Cayley graph perspective

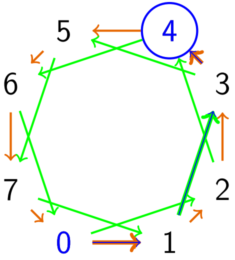
- Hash computation $\sim$ *walk* in the *Cayley graph*
- Example : $G = (\mathbb{Z}/8\mathbb{Z}, +)$, $S = \{1, 2\}$



$$m = 101$$
$$H(m) = 0 + 1 + 2 + 1 = 4$$

Cayley graph perspective

- Hash computation $\sim$ *walk* in the *Cayley graph*
- Example : $G = (\mathbb{Z}/8\mathbb{Z}, +)$, $S = \{1, 2\}$



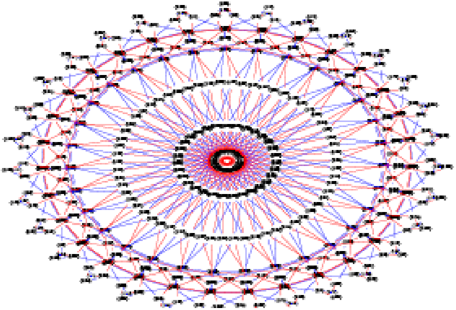
$m = 101$

$H(m) = 0 + 1 + 2 + 1 = 4$

- Preimage algorithm $\sim$ path-finding algorithm

Example : Tillich-Zémor [TZ94]

$$G = SL(2, \mathbb{F}_{2^n}), \qquad S = \left\{ A_0 = \begin{pmatrix} X & 1 \\ 1 & 0 \end{pmatrix}, A_1 = \begin{pmatrix} X & X+1 \\ 1 & 1 \end{pmatrix} \right\}$$



A hard (?) problem

- **Factorization problem in finite groups** :
  Given G, $g \in G$ and $S = \{s_0, ..., s_{k-1}\} \subset G$,
  find a short product $\prod s_{m_i} = g$

A hard (?) problem

- **Factorization problem in finite groups** :
  Given G, $g \in G$ and $S = \{s_0, \ldots, s_{k-1}\} \subset G$,
  find a short product $\prod s_{m_i} = g$
- Corresponds to finding preimages
- Similar problems for collision, second preimage

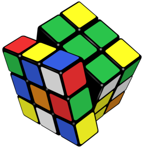
A hard (?) problem

- **Factorization problem in finite groups** :
  Given G, $g \in G$ and $S = \{s_0, ..., s_{k-1}\} \subset G$,
  find a short product $\prod s_{m_i} = g$

- Corresponds to finding preimages
- Similar problems for collision, second preimage

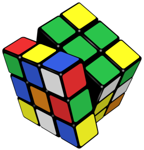
- Has this problem been sufficiently studied ?

Popular example : the Rubik's cube



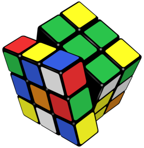
- Rubik's cube $\sim$ subgroup of all permutations of the corners, the central edge elements and their orientations

Popular example : the Rubik's cube



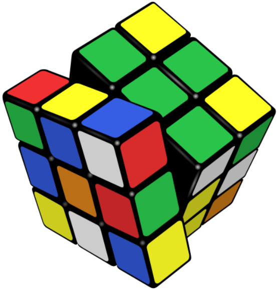
- Rubik's cube $\sim$ subgroup of all permutations of the corners, the central edge elements and their orientations
- Generated by the faces' rotations
- Neutral element $\sim$ Rubik's cube when solved

Popular example : the Rubik's cube

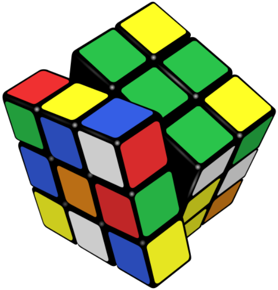


- Rubik's cube $\sim$ subgroup of all permutations of the corners, the central edge elements and their orientations
- Generated by the faces' rotations
- Neutral element $\sim$ Rubik's cube when solved
- Solution $=$ combination of the elementary permutations leading to the neutral element

Is Rubik hard enough ?



Is Rubik hard enough ?



Not really, but generalizations might be

Related problems

- **Babai's conjecture** [BS92]
  - *There is a constant c such that, for any non-Abelian finite simple group G, for all generator sets S, the diameter of the Cayley graph arising from G and S is smaller than $(\log|G|)^c$.*
  - Partial proofs by Helfgott, Tao, Bourgain,...

Related problems

- **Babai's conjecture** [BS92]
  - *There is a constant c such that, for any non-Abelian finite simple group G, for all generator sets S, the diameter of the Cayley graph arising from G and S is smaller than $(\log |G|)^c$.*
  - Partial proofs by Helfgott, Tao, Bourgain,...
  - Factoring problem $\sim$ constructive proof of the conjecture

Related problems

- **Babai's conjecture** [BS92]
  - *There is a constant c such that, for any non-Abelian finite simple group G, for all generator sets S, the diameter of the Cayley graph arising from G and S is smaller than $(\log |G|)^c$.*
  - Partial proofs by Helfgott, Tao, Bourgain,...
  - Factoring problem $\sim$ constructive proof of the conjecture

- **Expander graphs**
  - Cayley graphs tend to be good expanders
  - Expanders have a lot of applications [HLW06]
  - Traveling in those graphs will be useful, too

Is the problem hard enough ?

- **LPS hash function** [CGL07]
  - Collision and preimage attacks [TZ08,PLQ08]

Is the problem hard enough ?

- **LPS hash function** [CGL07]
  - Collision and preimage attacks [TZ08,PLQ08]
- **Tillich-Zémor hash function** [TZ94]
  - Collision and preimage attacks [GIMS11,PQ10]

Is the problem hard enough ?

- **LPS hash function** [CGL07]
  - Collision and preimage attacks [TZ08,PLQ08]
- **Tillich-Zémor hash function** [TZ94]
  - Collision and preimage attacks [GIMS11,PQ10]
- Other **particular parameters** broken

Is the problem hard enough ?

- **LPS hash function** [CGL07]
  - Collision and preimage attacks [TZ08,PLQ08]
- **Tillich-Zémor hash function** [TZ94]
  - Collision and preimage attacks [GIMS11,PQ10]
- Other **particular parameters** broken
- **General parameters** : work in progress

Some advices to Mary Stuart

- Use MACs or signatures to authenticate the messages
- Don't use MD5 !

Some advices to Mary Stuart

- Use MACs or signatures to authenticate the messages
- Don't use MD5 !
- Too risky to use hash functions from Cayley graphs

Some advices to Mary Stuart

- Use MACs or signatures to authenticate the messages
- Don't use MD5 !

- Too risky to use hash functions from Cayley graphs
- Working on generalizations of the Rubik's cube will be a funny and useful way to spend your time in prison
  - Expander graphs and their applications
  - Babai's conjecture
  - Cryptographic applications

Outline

Elliptic curve cryptography

Hash functions and the Rubik's cube

Side-channel attacks

An example of side-channel attack

- Choose a random bit $b \in \{0, 1\}$. Keep it perfectly secret.

An example of side-channel attack

- Choose a random bit $b \in \{0, 1\}$. Keep it perfectly secret.
- Let $x := b \times 123456789$. Keep this number secret.

An example of side-channel attack

- Choose a random bit $b \in \{0, 1\}$. Keep it perfectly secret.
- Let $x := b \times 123456789$. Keep this number secret.
- Let $y := x^2$. Keep this number secret.
  When you're done, raise a hand.

An example of side-channel attack

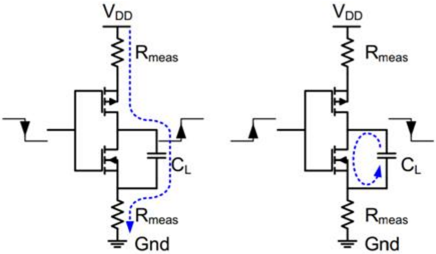
- Choose a random bit $b \in \{0, 1\}$. Keep it perfectly secret.
- Let $x := b \times 123456789$. Keep this number secret.
- Let $y := x^2$. Keep this number secret.
  When you're done, raise a hand.
- Let $z := 0 \times y$. Return z.

An example of side-channel attack

- Choose a random bit $b \in \{0, 1\}$. Keep it perfectly secret.
- Let $x := b \times 123456789$. Keep this number secret.
- Let $y := x^2$. Keep this number secret.
  When you're done, raise a hand.
- Let $z := 0 \times y$. Return z.

- From z only, I know nothing about b
- From **computing time**, I can guess b with a good probability.

CMOS inverter dynamic consumption



Charge vs. discharge of a CMOS inverter.

Figure credit : FX Standaert

$$P = C_L V_{DD}^2 P_{0 \to 1} f$$

If you can't get through it, go around it

- In most crypto algorithms, recovering the private key from the messages would require solving a very hard problem

If you can't get through it, go around it

- In most crypto algorithms, recovering the private key from the messages would require solving a very hard problem
- **Side-channel attacks** : use computing side information
  - Timing, computing power, electromagnetic variations, keyboard noise,...

If you can't get through it, go around it

- In most crypto algorithms, recovering the private key from the messages would require solving a very hard problem
- **Side-channel attacks** : use computing side information
  - Timing, computing power, electromagnetic variations, keyboard noise,. . .
- **Fault attacks** : induce faults during computation, deduce relevant information from the result
  - Alter memory
  - Skip some instructions

Square and Multiply algorithm (SM)

- In RSA, need to compute $\mathbf{g}^{\mathbf{d}}$ mod $\mathbf{n}$ where d is secret

Square and Multiply algorithm (SM)

- In RSA, need to compute $\mathbf{g^d}$ mod $\mathbf{n}$ where d is secret
- Modular exponentiations use **SM algorithm**

Square and Multiply algorithm (SM)

- In RSA, need to compute $\mathbf{g}^{\mathbf{d}}$ mod $\mathbf{n}$ where d is secret
- Modular exponentiations use **SM algorithm**
  1. Let $d = d_0 + d_1 2 + d_2 2^2 + ... + d_N 2^\ell$
  2. Let $h := 1$
  3. For $i := \ell, \ldots, 0$ do
  4. $\quad \mathbf{h} \leftarrow \mathbf{h}^2 \text{ mod } n$
  5. $\quad$ If $d_i = 1$ then
  6. $\quad\quad \mathbf{h} \leftarrow \mathbf{hg} \text{ mod } n$
  7. $\quad$ end if
  8. end for

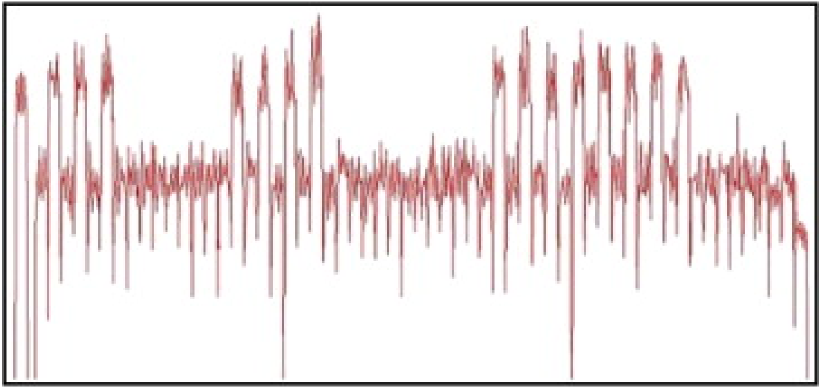
Square and Multiply algorithm (SM)

- In RSA, need to compute $\mathbf{g^d}$ mod $\mathbf{n}$ where d is secret
- Modular exponentiations use **SM algorithm**
  1. Let $d = d_0 + d_1 2 + d_2 2^2 + \ldots + d_N 2^\ell$
  2. Let $h := 1$
  3. For $i := \ell, \ldots, 0$ do
  4. $\quad \mathbf{h} \leftarrow \mathbf{h^2} \bmod n$
  5. $\quad$ If $d_i = 1$ then
  6. $\quad\quad \mathbf{h} \leftarrow \mathbf{hg} \bmod n$
  7. $\quad$ end if
  8. end for
- Always square, but multiply only when the bit is 1

Square and Multiply algorithm (SM)

- In RSA, need to compute $\mathbf{g}^{\mathbf{d}}$ mod $\mathbf{n}$ where d is secret
- Modular exponentiations use **SM algorithm**
  1. Let $d = d_0 + d_1 2 + d_2 2^2 + \ldots + d_N 2^\ell$
  2. Let $h := 1$
  3. For $i := \ell, \ldots, 0$ do
  4. $\quad\mathbf{h} \leftarrow \mathbf{h}^2 \bmod n$
  5. $\quad$If $d_i = 1$ then
  6. $\quad\quad\mathbf{h} \leftarrow \mathbf{hg} \bmod n$
  7. $\quad$end if
  8. end for
- Always square, but multiply only when the bit is 1
- What is the power consumption ?

A power attack against SM

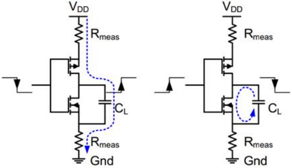


Correlation power attack

- **Divide and conquer** : succesively recover key bytes

Correlation power attack

- **Divide and conquer** : succesively recover key bytes
- **Leakage model**
  - Hamming distance
  - Hamming weight

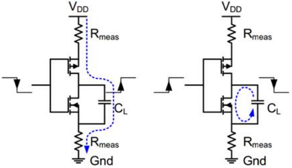


Charge vs. discharge of a CMOS inverter.

Figure credit : FX Standaert

Correlation power attack

- **Divide and conquer** : succesively recover key bytes
- **Leakage model**
  - Hamming distance
  - Hamming weight



Charge vs. discharge of a CMOS inverter.

Figure credit : FX Standaert

- **Correlation attack**
  - Make a guess on a key byte
  - Deduce Hamming weight (variations) of the registers
  - Correlate with the power trace(s)

Variations

- **Signal preprocessing** to reduce noise

Variations

- **Signal preprocessing** to reduce noise
- **Dimensionality reduction** to select points on the traces

Variations

- **Signal preprocessing** to reduce noise
- **Dimensionality reduction** to select points on the traces
- If another device available, build **leakage templates** to improve the leakage model

Variations

- **Signal preprocessing** to reduce noise
- **Dimensionality reduction** to select points on the traces
- If another device available, build **leakage templates** to improve the leakage model
- Other statistics or **machine learning** tools to identify the right key candidate

Variations

- **Signal preprocessing** to reduce noise
- **Dimensionality reduction** to select points on the traces
- If another device available, build **leakage templates** to improve the leakage model
- Other statistics or **machine learning** tools to identify the right key candidate
- **Brute-force** to eliminate last wrong key candidates

Countermeasures

- Physical countermeasures
  - Physical and chemical shields
  - Noise addition
  - Dual-rail logic styles
  - . . .

Countermeasures

- Physical countermeasures
  - Physical and chemical shields
  - Noise addition
  - Dual-rail logic styles
  - ...
- Algorithmic countermeasures
  - Dummy operations
  - Noise addition
  - Masking
  - Shuffling
  - ...

Fresh rekeying

- Because of noise, side-channel attacks typically require **many traces from the same key**

Fresh rekeying

- Because of noise, side-channel attacks typically require **many traces from the same key**
- Idea : build new algorithms/protocols for which the **key is frequently updated** [PSPMY08,MPRRS11,...]

Fresh rekeying

- Because of noise, side-channel attacks typically require **many traces from the same key**
- Idea : build new algorithms/protocols for which the **key is frequently updated** [PSPMY08,MPRRS11,...]
- If possible, build them from standard algorithms

A general advice to Mary Stuart

- Beware that even a secure algorithm can become unsecure if badly implemented

A general advice to Mary Stuart

- Beware that even a secure algorithm can become unsecure if badly implemented
- Include appropriate side-channel counter-measures in your favorite crypto computing machine

Outline

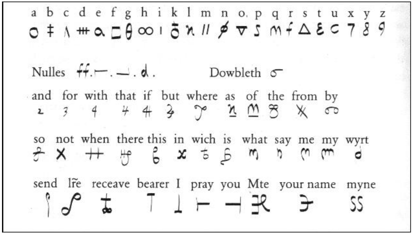
Elliptic curve cryptography

Hash functions and the Rubik's cube

Side-channel attacks

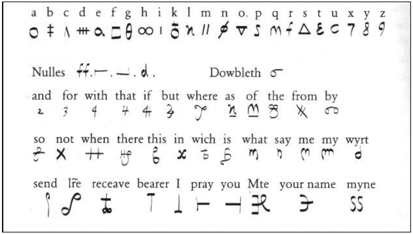
“Cipher of Death”

- Mary Stuart **didn’t use good crypto**
- Her code was broken by Thomas Phelippes



“Cipher of Death”

- Mary Stuart **didn't use good crypto**
- Her code was broken by Thomas Phelippes



- Walsingham sent her a **fake message** asking confirmation of her commitment ; she answered
- Mary sentenced to death and executed on Feb 8th, 1587

Conclusion

- **We all need a good cryptographer**
  - More than military and government usage today
  - Private communications, ATMs, e-banking, e-voting,. . .

Conclusion

- **We all need a good cryptographer**
  - More than military and government usage today
  - Private communications, ATMs, e-banking, e-voting,. . .
- Challenges for the good (?) guy
  - Make algorithms fast, tiny and secure
  - New crypto applications

Conclusion

- **We all need a good cryptographer**
  - More than military and government usage today
  - Private communications, ATMs, e-banking, e-voting,. . .
- Challenges for the good (?) guy
  - Make algorithms fast, tiny and secure
  - New crypto applications
- Challenges for the bad (?) guy
  - New algorithms for hard problems (ECDLP,. . . ...)
  - Perform huge cryptanalysis tasks
  - New side-channel attacks

Credits

- The first chapter of Simon Singh's *Code Book* clearly inspired the introduction of this talk.

References

- [B92] L Babai, A Seress, *On the diameter of permutation groups.*
- [CGL09] D Charles, E Goren, K Lauter, *Cryptographic Hash Functions from Expander Graphs.*
- [FPPG12] JC Faugère, L Perret, C Petit, G Renault, *Improving the complexity of index calculus for elliptic curves over binary fields.*
- [GIMS11] M Grassl, I Ilic, S Magliveras, R Steinwandt, *Cryptanalysis of the Tillich-Zémor hash function.*
- [H08] H Helfgott, *Growth and generation in* $SL2(Z/pZ)$.

References

- [MPRRS11] M Medwed, C Petit, F Regazzoni, M Renauld, FX Standaert, *Fresh Re-keying II : Securing Multiple Parties against Side-Channel and Fault Attacks.*
- [PLQ08] C Petit, K Lauter, JJ Quisquater, *Full Cryptanalysis of LPS and Morgenstern Hash Functions.*
- [PSPMY08] C Petit, FX Standaert, O Pereira, T Malkin, M Yung, *A block cipher based pseudo random number generator secure against side-channel key recovery.*
- [PQ10] C Petit, JJ Quisquater. *Preimages for the Tillich-Zémor hash function.*

References

- [PQ12] C Petit, JJ Quisquater. *On polynomial systems arising from a Weil descent.*
- [S99] S Singh, *The Code Book.*
- [TZ94] JP Tillich, G Zémor. *Group-theoretic hash functions.*
- [TZ08] JP Tillich, G Zémor. *Collisions for the LPS Expander Graph Hash Function.*

RSA encryption algorithm

- **Key generation**
  - Private key is a couple of primes (p, q).
  - Public key is (n, e) where $n = pq$.

RSA encryption algorithm

- **Key generation**
  - Private key is a couple of primes (p, q).
  - Public key is (n, e) where $n = pq$.
- **Encryption**
  - Given a message m, compute $c := m^e \bmod n$

RSA encryption algorithm

- **Key generation**
  - Private key is a couple of primes (p, q).
  - Public key is (n, e) where $n = pq$.
- **Encryption**
  - Given a message m, compute $c := m^e \bmod n$
- **Decryption :**
  - Knowing (p, q), compute d such that $ed = 1 \bmod (p-1)(q-1)$
  - Compute $c^d \bmod n = m^{ed} \bmod n = m \bmod n$

RSA encryption algorithm

- **Key generation**
  - Private key is a couple of primes (p, q).
  - Public key is (n, e) where $n = pq$.
- **Encryption**
  - Given a message m, compute $c := m^e \bmod n$
- **Decryption :**
  - Knowing (p, q), compute d such that $ed = 1 \bmod (p-1)(q-1)$
  - Compute $c^d \bmod n = m^{ed} \bmod n = m \bmod n$

- Everybody can encrypt, but private key needed to decrypt
- Computing (p, q) from the public key is the **integer factorization problem**

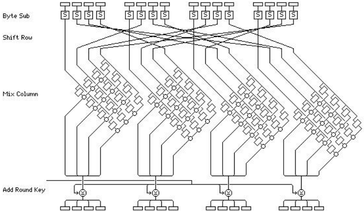
Advanced Encryption Standard (AES)

- Algorithm Rijndael (Vincent Rijmen and Joan Daemen)
- Selected in 2001 after public competition
- Replaced previous standards DES and 3-DES

Advanced Encryption Standard (AES)

- Algorithm Rijndael (Vincent Rijmen and Joan Daemen)
- Selected in 2001 after public competition
- Replaced previous standards DES and 3-DES

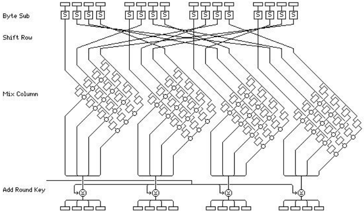
- Key size of 128, 192, or 256 bits
- Block size 128 bits
- 10, 12, or 14 rounds



Advanced Encryption Standard (AES)

- Algorithm Rijndael (Vincent Rijmen and Joan Daemen)
- Selected in 2001 after public competition
- Replaced previous standards DES and 3-DES

- Key size of 128, 192, or 256 bits
- Block size 128 bits
- 10, 12, or 14 rounds



- **Assumption** : AES good pseudo-random permutation

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$

- **Relation search**
  - Choose random $a, b \in \{1, \ldots, p-1\}$
  - Compute $r := g^a h^b \bmod p$

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$

- **Relation search**
  - Choose random $a, b \in \{1, \ldots, p-1\}$
  - Compute $r := g^a h^b \bmod p$
  - If all factors of r are $\leq B$, store a relation $[a]g + [b]h = \sum_{p_i \in \mathcal{F}_B} [e_i] p_i$

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**

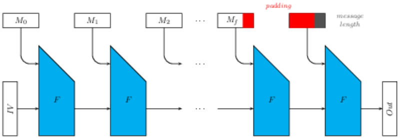
$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$

- **Relation search**
  - Choose random $a, b \in \{1, \ldots, p-1\}$
  - Compute $r := g^a h^b \bmod p$
  - If all factors of r are $\leq B$, store a relation $[a]g + [b]h = \sum_{p_i \in \mathcal{F}_B} [e_i] p_i$
- **Linear algebra** modulo $p-1$ on the relations

Example : a naive index calculus for $\mathbb{F}_p^*$

- DLP : given $g, h \in \mathbb{F}_p^*$, find k such that $h = g^k$
- Factor basis made of **small "primes"**
$$\mathcal{F}_B := \{\text{primes } p_i \leq B\}$$
- **Relation search**
  - Choose random $a, b \in \{1, \ldots, p-1\}$
  - Compute $r := g^a h^b \bmod p$
  - If all factors of r are $\leq B$, store a relation $[a]g + [b]h = \sum_{p_i \in \mathcal{F}_B} [e_i] p_i$
- **Linear algebra** modulo $p-1$ on the relations
- For $B \approx \exp((\log p)^{1/2})$, **subexponential complexity**

A very high-level look at SHA-1



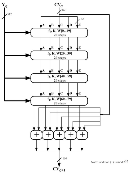


Figure 9.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

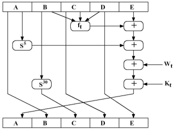
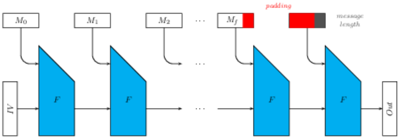


Figure 9.6 Elementary SHA Operation (single step)

- Most hash functions have a similar structure

A very high-level look at SHA-1



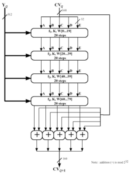


Figure 9.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

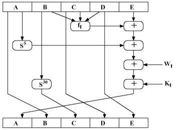


Figure 9.6 Elementary SHA Operation (single step)

- Most hash functions have a similar structure
- Security : various heuristic arguments